AKC 攻击研究: 攻击方式、转换算法和实例分析①

麻 婧[1,2], 张文辉[1]

[1](中国科学院软件研究所 计算机科学国家重点实验室, 北京 100190)
[2](中国科学院大学, 北京 100190)

摘 要: 攻击者获取某主体(actor)的长期私钥后, 利用该私钥伪装成其他主体欺骗 actor 或获取保密信息的行为被称为 AKC(Actor Key Compromise)攻击. 除密钥交换协议外, AKC 攻击在其他类型的协议研究中较少受关注. 本文强调了 AKC 攻击问题的重要性, 并对其攻击方式和应对策略进行系统研究. 通过实验总结出 4 类 AKC 攻击方式, 并对应提出 3 类抵制 AKC 攻击的协议模型和设计原则. 在此基础上, 给出了将一般协议转换为 AKCS 协议(在 AKC 攻击下保持安全性质的协议)的启发式算法. 在实例分析中, 将算法应用在 Email、SET、Kerberos 等协议上. 实验表明, 上述协议受 AKC 攻击, 但在算法的转换下, 协议不再受 AKC 攻击影响.

关键词: 安全协议; AKC 攻击; 安全性质; 协议转换

Research on AKC Attack: Attack Pattern, Transformation Algorithm and Case Study

MA Jing[1,2], ZHANG Wen-Hui[1]

[1](Institute of Software, Chinese Academy of Sciences, State Key Laboratory of Computer Science, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: After the fact that an adversary obtains an actor's long-term secret key, the adversary may impersonate other actors or obtain secrete information with the key. This kind of attack is called AKC (Actor Key Compromise) attack. Except for key exchange protocols, not much attention has been paid on other types of security protocols in the research of AKC attacks. In this paper, we consider this AKC problem and provide systematic analysis of AKC attacks, its attack patterns and countermeasures. Based on experimental analysis, this paper classifies four AKC attack patterns, and three corresponding protocol models and design principles to protect against AKC attacks. Based on these models and principles, it proposes a heuristic algorithm that transforms a protocol into an AKCS one (that keeps security property under AKC attacks). As case studies, it apply the algorithm on protocols including Email, SET and Kerberos. The results show that these protocols are vulnerable to AKC attacks, but after the transformation by the algorithm, they are no longer vulnerable to such attacks.

Key words: security protocol; AKC attack; security property; protocol transformation

在公钥加密体系(PKI)下有一类有趣的现象: 当某个主体(actor)的长期私钥泄露后, 攻击者不但可以伪装成 actor 向其他主体发送消息, 还可以反过来伪装成其他主体欺骗 actor. 我们称这类攻击为 AKC 攻击: 攻击者获取 actor 的长期私钥并利用它窃听、拦截、篡改消息, 使得 actor 的单边安全性质(如数据保密性、完整性、认证性)被破坏, 即便与 actor 通信的其他主体的长期私钥是保密的. 本文中 actor 表示受 AKC 攻击的主体. 在实际攻击中, actor 可以是邮件服务器, 电商中的客户, 或某个等待接入局域网的计算机.

研究 AKC 攻击问题具有重要意义. 很多协议在 Dolev-Yao 攻击者模型下是安全的, 但在 AKC 模型下无法满足 actor 单边安全性质. 在有越来越强攻击者存在的网络环境中, 这类协议的安全性较弱. 例如 OpenSSL 爆出的 Heartbleed 漏洞中, 攻击者可获取服务器运行内存, 从而访问服务器长期私钥等敏感数据. 利用该

① 基金项目:国家自然科学基金(61272135)
收稿时间:2016-01-18;收到修改稿时间:2016-03-08 [doi:10.15888/j.cnki.csa.005375]

私钥, 攻击者可以在服务器与客户端之间发动被动中间人攻击, 解密当前的或已存储的传输数据.

但目前阶段对 AKC 攻击的研究比较局限. 一类研究只针对密钥交换类协议, 称之为 Key Compromise Impersonation(KCI) [1]攻击: 攻击者获取 actor 长期私钥后伪装成其他主体与 actor 协商会话密钥. KCI 攻击只是 AKC 攻击的一小部分实例. 另一类研究以 Cas Cremers 为代表, 对 AKC 攻击下的安全协议的语义、安全性质建立模型, 并给出预防 AKC 攻击的一般性结论[2]. 然而, 这些结论对设计 AKCS 协议并不足够. 并且没有一个一般性的方法将 Dolev-Yao 模型下安全的协议转换为 AKCS 协议.

抛开协议类型的限制, 本文对 AKC 攻击的机理和应对策略进行了系统研究. 研究的协议范围覆盖两方、多方参与协议, 协议类型包含认证、公平交换(Email)、电子商务协议等. 本文贡献如下: (1)总结 AKC 攻击方式并分析其机理; (2)针对每类攻击方式, 给出对应的 AKCS 协议模型和设计原则; (3)提出将一般协议转换为 AKCS 协议的转换算法; (4)使用 Scyther[3]工具, 发现多个协议中存在的 AKC 攻击, 并应用算法转换为 AKCS 协议.

1 预备知识

本文的研究基于 Cas Cremers 提出的安全协议框架[4]. 在该框架中, 协议主体有两类: 诚实主体和攻击者. 前者具备发送、接受消息的能力. 后者具备获取长期私钥, 监听、拦截和发送消息的能力. 消息由 *Term* 组成, 其定义如下:

$$Term ::= Agent \mid Role \mid Fresh \mid Var \mid Fresh^{\#TID} \mid Var^{\#TID}$$
$$\mid Func(Term) \mid (Term, Term) \mid \{Term\}_{Term}$$
$$\mid sk(Term) \mid pk(Term) \mid k(Term, Term)$$

其中 *Role* 表示参与者角色, *Agent* 表示运行主体, *Fresh* 表示新生成的随机数, *Var* 表示存储接受信息的变量, *TID* 表示运行的标识号, *Func* 表示函数, *sk*、*pk* 表示长期私钥和公钥, *k* 表示对称密钥. 规定本文中 *Func* 只有两种: 在协议消息中使用的哈希函数 *h* 和在安全声明中使用的哈希函数 *h'*. 哈希函数具有信息校验和数字签名的功能. 定义 Accessible 关系 $\prec$: 对任意 $T1 \in Term$、$T2 \in Term$, 有 $T1 \prec (T1, T2)$, $T2 \prec (T1, T2)$, $T1 \prec \{T1\}_{T2}$. 本文中“*X* 的出现”表示在消息中 Accessible 位置出现的 *Term X*.

协议由一系列事件(Event)序列构成, 定义如下:

$$AgentEvent ::= create(Role, Agent) \mid send_{label}(Term)$$
$$\mid recv_{label}(Term) \mid claim_{label}(a,Claim[,Agent][,Term])$$
$$AdversaryEvent ::= LKR(Agent) \mid send_{label}(Term)$$
$$Event ::= AgentEvent \mid AdversaryEvent$$

其中 *AgentEvent* 表示诚实主体发生的事件, *AdversaryEvent* 表示攻击者发生的事件. *create* 表示创建角色实例. *send(recv)* 表示发送(接收)消息, 在 *send(recv)*事件中要指明发送者和接收者. 使用 *label* 对事件标号. *claim* 表示安全性质声明, 其中 *a* 表示 *actor*. *LKR(Agent)*表示攻击者获取 *sk(Agent)*, 本文中有时用 *LKR(Role)*表示攻击者在运行中获取 *sk(Role)*的实例. *Claim* 表示具体的安全性质, 本文中有 4 种:

1) *Secret*: 声明为 $claim_L(R, Secret, t)$, 表示在 *LKR(R)* 下攻击者无法获取或推断出 *t*, 保证了 *t* 的保密性.

2) *Commit*: 声明为 $claim_L(R, Commit, R', t)$, 表示在 *LKR(R)*下, *claim* 事件之前存在事件 $send_L(R', R, M)$ 且 $t \in M$. *Commit* 性质使 *R* 确认 *t* 来自 *R'*发送的消息中, 保证了 *t* 的完整性.

3) *Nisynch*: 声明为 $claim_L(R, Nisynch)$, 表示在 *LKR(R)*下, *claim* 事件之前的所有 *send(recv)*事件都能被正确的 *Agent* 以正确的顺序和内容执行. *Nisynch* 性质保证 *actor* 接收到的信息都满足 *Commit* 性质, 因此具备较强认证性.

4) *Niagree*: 声明为 $claim_L(R, Niagree)$. 其含义与 *Nisynch* 类似, 但在事件的执行顺序上不做要求, 只要求声明之前的所有 *send(recv)*事件都能以正确的内容被执行.

当协议交互的双方仅使用对称密钥加密信息时, 该协议一定受到 AKC 攻击[2]. 因此本文中限制协议只能使用 4 种加密形式: 长期公钥、长期私钥、临时随机数、哈希函数. 并且规定任意一方主体不得向其他主体发送自己的长期私钥.

2 AKC攻击方式

AKC 攻击的后果有三种: (1)获取协议中的保密信息; (2)破坏协议会话的对应性; (3)破坏协议的可用性. 不同后果是由不同攻击方式引起的. 通过分析 AKC 攻击方式, 可以找到应对攻击的策略.

本文总结的 4 类攻击方式非首次提出. 只是在其他对攻击方式的研究中, 多数以 Dolev-Yao 模型为假

设前提. 但实验中发现, 很多在 Dolev-Yao 下验证为安全的协议在 AKC 模型下存在攻击. 因此本文针对 AKC 模型总结了 4 类攻击方式, 攻击者在这 4 类攻击中都利用了 sk(actor).

2.1 数据保密性攻击

AKC 模型下数据保密性攻击是指: 攻击者利用 actor 长期私钥对消息监听并解密, 从中直接获取或计算出保密性数据, 从而破坏 Secret 性质.

以 Bilateral Key Exchange(BKE)[5]协议为例:

① $I \rightarrow R : \{ni, I\}_{pk(R)}$

② $R \rightarrow I : \{h(ni), nr, R, kir\}_{pk(I)}$

③ $I \rightarrow R : \{h(nr)\}_{kir}$

BKE 协议利用 *I*、*R* 产生的随机数(*ni*、*nr*)验证彼此的公钥信息, 并产生临时会话密钥 *kir*. 因此双方要求 kir 保密($claim_1(I, Secret, kir)$、$claim_2(R, Secret, kir)$). 在 *LKR(I)*下前者不成立: 攻击者获取 *sk(I)*后可以解密消息 2, 从而获得 *kir*.

2.2 中间人攻击

中间人攻击是一种通过拦截正常网络数据, 并对数据进行篡改转发, 而通信双方毫不知情的攻击手段. 在 AKC 模型下, 攻击者因为具有解密消息和伪造签名的能力, 拥有更多实施中间人攻击的机会.

AKC 模型下的中间人攻击分为两类: 一类利用私钥解密消息, 另一类利用私钥伪造签名. 这两类都会利用私钥篡改转发消息.

首先使用 1.1 中 BKE 协议说明第一类攻击. BKE 协议要求 *I*、*R* 能确认对方产生的随机数($claim_3(Alice, Nisynch)$、$claim_4(Bob, Nisynch)$). 但在 AKC 模型下存在如下攻击:

① $Alice \rightarrow Bob : \{ni, Bob\}_{pk(Bob)}$

② $Bob \rightarrow D_{Alice} : \{h(ni), nr, Alice, kir\}_{pk(Alice)}$

③ D_{Alice} 使用 *sk(Alice)*解密消息 2, 生成 *nr'*

④ $D_{Alice} \rightarrow Alice : \{h(ni), nr', Alice, kir\}_{pk(Alice)}$

⑤ $Alice \rightarrow D_{Alice} : \{h(nr')\}_{kir}$

其中 *Alice*、*Bob* 表示角色 *I*、*R* 的执行主体, D_{Alice} 表示攻击者伪装成 *Alice* 收发消息. 在 *LKR(Alice)*下, 攻击者利用 *sk(Alice)*解密消息 2, 并使用自己产生的随机数 *nr'*发送给 *Alice*, 而 *Alice* 不知道自己接收的 *nr'*是由攻击者产生的, 因此 *Nisynch* 性质不满足.

其次使用 PKMv2-RSA[6]协议说明第二类攻击. PKMv2-RSA 是 IEEE 802.16 中的一个子协议, 完成基站(*MS*)与移动电台(*BS*)的相互认证并产生共享密钥(*prepak*). 因此除了要求 *prepak* 保密外, 要保证每条消息的数据完整性($claim_3(MS, Nisynch)$、$claim_4(BS, Nisynch)$). 协议过程如下:

① $MS \rightarrow BS : \{msrand, said, MS\}_{sk(MS)}$

② $BS \rightarrow MS : \{msrand, bsrand, \{prepak, MS\}_{pk(MS)}, BS\}_{sk(BS)}$

③ $MS \rightarrow BS : \{bsrand\}_{sk(MS)}$

其中 *msrand*、*said*、*bsrand* 是双方用于确认对方身份而产生的随机数. 在 *LKR(MS)*下 *Nisynch* 性质不满足: 攻击者可拦截消息 1, 利用 *sk(MS)*和自己产生的随机数 *said'*构造$\{msrand, said', MS\}_{sk(MS)}$并发送给 *BS*, 使 *MS*、*BS* 在 *said* 上没达成一致.

2.3 交错攻击

交错攻击将协议的多个运行实例安排为以交织的方式执行, 其危害是影响认证性质或协议可用性. 例如 STS 协议存在的交错攻击会导致拒绝服务(DoS).

AKC 模型下交错攻击分两类: 一类针对 two-party 协议(如图 1), 一类针对 multi-party 协议(如图 2). 图中 *A(i)#1* 表示主体 A 在运行 1 中是角色 i 的实例. 交错攻击中存在着对消息的篡改转发, 因而交错攻击包含了中间人攻击.

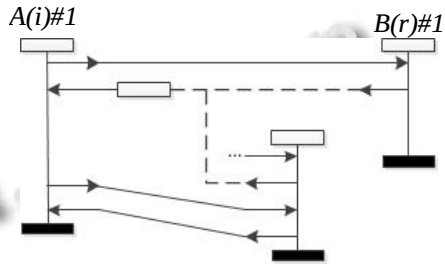

图 1　two-party 协议交错攻击

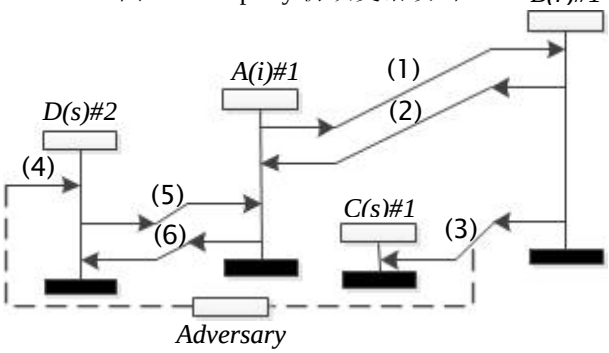

图 2　multi-party 协议交错攻击

首先讨论第一类攻击. 图 1 中协议存在两个并发运行 1、2, 其中运行 1 的执行主体是 *A*、*B*, 运行 2 的

执行主体是 *C*、*D*(为图示简洁省略 *D*). *A*、*B*(*D*、*C*)分别在运行 1(2)中作为角色 *i*、*r* 的实例. 攻击者利用 *sk(A)*解密消息(2)、(3), 并构造消息(4)发送给 *A*. 此时 *A* 获得了 *C* 的信息并开始与 *C* 对话. 在协议结束时, *A* 不知道后期是在与 *C* 对话, 而以为自己一直与 *B* 交互. 此类攻击常见于有多次交互的 two-party 协议中. 如 1.1 中 BKE 协议存在交错攻击: 在 *LKR(I)*下存在两个运行 1、2, 攻击者利用 *sk(I)*解密两个运行中的消息 2, 构造$\{hash(ni), nr\#1, R, kir\#2\}_{pk(I)}$发送给 *I*, 使得运行 1 中的 *I*、*R* 在 *kir* 上没达成一致.

其次讨论第二类攻击. 由于 multi-party 协议中消息的传送路径有多种可能, 图 2 仅表示其中一种可能. 由图中可以看出: *A*、*B*、*C* 作为运行 1 中 *i*、*r*、*s* 的实例. *A*、*B* 先完成一轮交互(1)(2), 之后 *B* 向 *C* 发送(3)并被攻击者拦截. 攻击者利用 *sk(C)*解密(3)并构造(4)转发给 *D*(运行 2 中 *s* 的实例), 此后 *A*、*D* 完成会话(5)(6). 协议结束时, *D* 以为自己与运行 2 中的 *i* 完成会话, 而 *A* 以为自己与 *C* 完成会话. 此类攻击可见于 Splice/AS[7]等协议中.

2.4 角色混淆攻击

角色混淆攻击是指, 协议的某一方主体对于其他参与者的身份存在错误的假设, 从而破坏了协议的认证性质. 本文使用 ISO/IEC 9798 标准下的一个认证协议说明 AKC 下的角色混淆攻击. 协议过程如下:

① $A \rightarrow B : Cert(A), RA, Text1$

② $B \rightarrow A : Cert(B), RB, Text2$

③ $B \rightarrow A : RB, RA, A, Text6, \{RB, RA, A, Text5\}_{sk(B)}$

④ $A \rightarrow B : RA, RB, B, Text5, \{RA, RB, B, Text3\}_{sk(A)}$

其中 *RA*、*Text1* 等是双方产生的随机数. 协议要求 *A*、*B* 在 *RA*、*RB* 和 *Text5* 上达成一致. 该协议在 AKC 模型下的角色混淆攻击如图 3 所示.

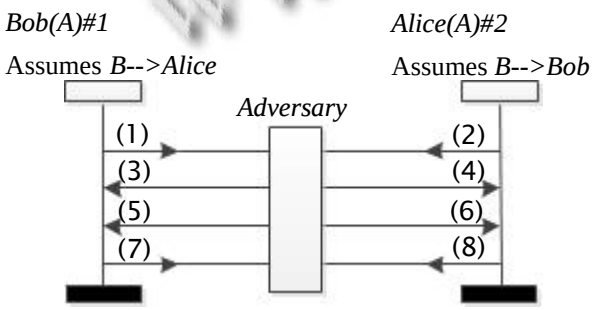

图 3 ISO/IEC 9798 3-5 协议角色混淆攻击

图 3 中 *Alice* 和 *Bob* 都作为角色 *A*, 并认为对方是 *B*. 攻击者利用 *sk(Bob)*在两者之间篡改转发消息. 协议结束时, *Alice* 和 *Bob* 都以为自己与角色 *B* 的主体完成了会话, 但事实上没有以角色 *B* 运行的主体. 因此 *claim(A, Commit, B, RA, RB, Text5)* 和 *claim(B, Commit, A, RA, RB, Text5)*性质不成立. 在 multi-party 协议中常见此类攻击.

3 AKCS协议模型和设计原则

第 2 节中给出了 4 类 AKC 攻击方式, 分析这 4 类攻击的本质原因, 可以找到解决 AKC 攻击的方法. 基于此分析, 本节中给出了若干 AKC 攻击下能保持安全性质的协议模型, 并提出 3 条协议设计原则.

3.1 **Secret** 模型

Secret 模型解决了数据保密性攻击的问题. 此类攻击的本质原因是, 加密方式单一导致消息容易被破解. 本文给出了 4 个在 AKC 下保持 Secret 性质成立的协议模型(如图 4-图 7). 其中模型 1 由 Cas Cremer 提出[2], 该模型在 AKC 攻击下保持单边的 Secret 性质. 本文对模型 1 进行了扩展, 产生模型 2、3、4, 且后两个模型在 LKR(R)和 LKR(R')下均成立, 具有更强的安全性质. 在足够的约束以及假设下, 这 4 个模型具备理论上的正确性, 但由于篇幅所限, 本文不给出证明.

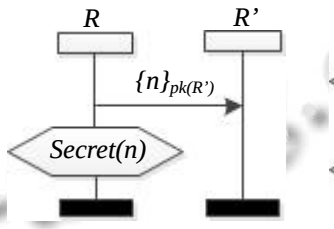

图 4 模型 1

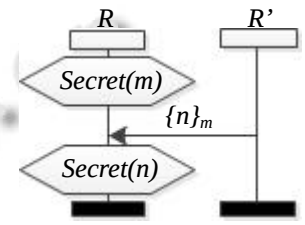

图 5 模型 2

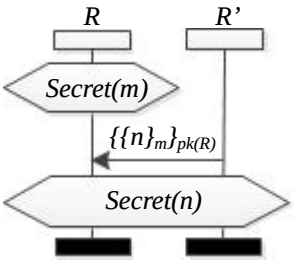

图 6 模型 3

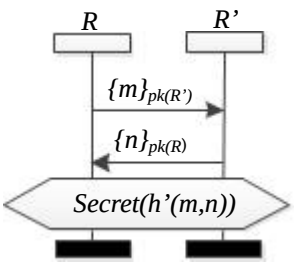

图 7 模型 4

模型 1: 在 *R* 与 *R'*的会话中, 若 *n* 的第一次出现是在 *R* 发送给 *R'*的消息中, 则将此处的 *n* 以 *pk(R')*加密, 并将其余出现用哈希函数 *h* 加密, 这样能保证 *LKR(R)* 下 *n* 的 *Secret* 性质. 在 multi-party 协议中, *n* 必须是由

R 产生的随机数, 且其他主体向 *R* 发送 *n* 时要使用 *h* 加密. 模型 1 的正确性在[2]中给出了证明.

模型 2: 在 *R* 与 *R'*的会话中, 若 *n* 的第一次出现是在 *R'*发送给 *R* 的消息中, 则选择一个 *m*, 将此处的 *n* 用 *m* 加密, 其余出现用 *h* 加密. 其中 *m* 需要满足 *LKR(R)*下的 *Secret* 性质(参考模型 1). 在 multi-party 协议中, *n* 必须是由 *R'*产生的随机数, 且其他主体向 *R* 发送 *n* 时要用 *h* 加密. 模型 2 正确性说明: 在 *LKR(R)*下 *m* 是保密的, 因此使用 *m* 加密 *n* 可以保证 *n* 的保密性.

模型 3: 是模型 1、2 的结合, 在模型 2 的基础上给第二条消息增加 *pk(R)*加密, 相当于采用了双层加密(公钥+临时随机数)方法, 使之保持双边 *Secret* 性质. 模型 3 正确性说明: 在 *LKR(R)*下 *m* 是保密的, 根据模型 2 可保证 *n* 的保密性; 在 *LKR(R')*下, 根据模型 1 可保证$\{n\}_m$的保密性, 进而保证 *n* 的保密性.

2.2 中对 BKE 协议的保密性攻击可用模型 3 预防: 将消息 2 修改为$\{h(ni), nr, R, \{kir\}_{ni}\}_{pk(I)}$. 其中 *ni* 对应于模型 3 中 *m*, *kir* 对应于 *n*. 由于 *ni*、*kir* 的后续出现都使用了哈希函数加密, 符合模型 3 约束条件, 因而满足 *LKR(I)*、*LKR(R)*下 *kir* 的 *Secret* 性质.

模型 4: 是模型 1 的变形, 用于建立 *R*、*R'*间的会话密钥 *h'(m,n)*. 这里对 *m*、*n* 的要求同模型 1. 模型 4 正确性说明: 在 *LKR(R)*下 *m* 是保密的, 因此攻击者无法构造 *h'(m,n)*; 在 *LKR(R')*下同理.

模型 4 可应用于 Needham-Schroeder-Low(NSL)[8]协议的转换中. NSL 协议中 *I*、*R* 互相交换信息并产生会话密钥 *ni*、*nr*. 在 AKC 攻击下, *ni*、*nr* 的 *Secret* 性质不满足. 图 8、9 展示了转换前、后的 NSL 协议:

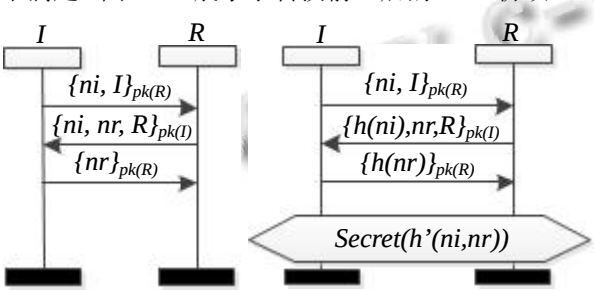

图 8　NSL 协议　　图 9　应用模型 4 的 NSL 协议

应用模型 4 将消息 2、3 中的 *ni*、*nr* 用 *h* 加密, 并使用 *h'(ni,nr)*作为 *I*、*R* 共享的会话密钥. 转换后的协议满足 *LKR(I)*、*LKR(R)*下 *K* 的 *Secret* 性质.

哈希加密原则: 消息中若包含接收方已知的信息 *N*, 为避免数据保密性攻击, 应以哈希函数加密的形式传送 *N*. 模型 1~4 中均使用了该原则.

3.2 Commit 模型

Commit 模型解决中间人攻击和 two-party 协议交错攻击问题. 这两类攻击的本质原因是, AKC 攻击下没能对接收到的信息确认其来源. 本文给出了 5 个在 AKC 攻击下保持 *Commit* 性质成立的协议模型(如图 10-14). 其中模型 5、6 由 Cas Cremer 提出[2]. 类似 3.1, 本文对模型 5、6 进行扩展, 产生模型 7、8、9, 且后两个模型在 *LKR(R)*、*LKR(R')*下均成立, 具有更强的安全性质.

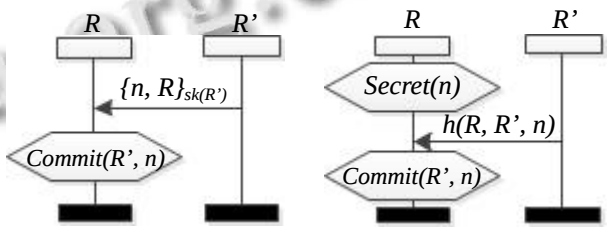

图 10　模型 5

图 11　模型 6

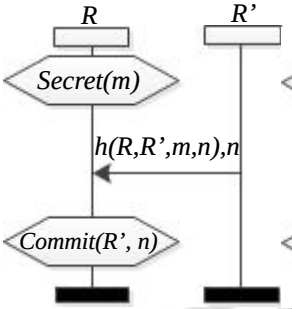

图 12　模型 7

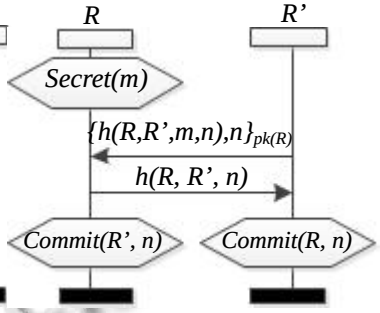

图 13　模型 8

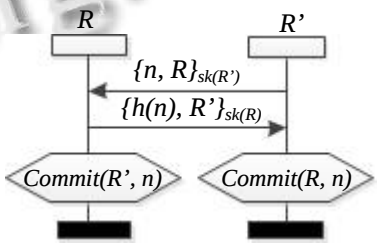

图 14　模型 9

模型 5: 若 *R'*产生随机数 *n*, 并将 *n* 用 *sk(R')*加密发送给 *R*, 则可保证 *R* 单边确认 *n*. 模型 5 的正确性在[2]中给出了证明.

模型 6: 若 *R'*发送给 *R* 一个 *LKR(R)*下保密的随机数 *n*, 并将其用哈希函数加密, 则可保证 *R* 单边确认 *n*. 模型中 *Secret(n)*的构造可参考 3.1 中模型. 模型 6 的正确性在[2]中给出了证明.

模型 7: 若 *R'*产生随机数 *n*, 并将 *n* 与 *LKR(R)*下保

密的随机数 *m* 一起用哈希加密传送给 *R*, 可保证 *R* 单边确认 *n*. 在应用中认为$\{R, R', n\}_m$与 *h(R, R', m, n), n* 是等效的. 模型 7 正确性说明: 在 *LKR(R)*下 *m* 是保密的, 攻击者无法构造 *h(R, R', m, n)*, 从而保证 *R* 对 *n* 的 *Commit* 性质.

模型 8: 是模型 6、7 的结合, 其中 *n* 是由 *R'*产生的随机数, 模型 8 保证双边对 *n* 的 *Commit* 性质. 模型 8 正确性说明: 在 *LKR(R)*下, 由模型 7 可得到 *R* 对 *n* 的 *Commit* 性质; 在 *LKR(R')*下, 由模型 1 可得到 *R'*对 *n* 的 *Secret* 性质, 并根据模型 6 得到 *R'*对 *n* 的 *Commit* 性质.

2.2 中对 BKE 协议的中间人攻击可用模型 8 解决. 在应用 3.1 中 Secret 模型的基础上, 将消息 2 修改为$\{h(I, R, ni, nr), nr, R, \{kir\}_{ni}\}_{pk(I)}$, 将消息 3 修改为$\{h(I, R, nr)\}_{kir}$(*ni* 对应于模型 8 中 *m*, *kir*、*nr* 对应于 *n*). 修改后的协议仍存在 2.3 中交错攻击, 这是由于消息 2 中 *nr*、*kir* 没有在同一个哈希函数中确认.

共同确认原则: 应用 Commit 模型时, 如果一条消息中有多个随机数需要确认, 则应保证每两个随机数至少有一次在同一个哈希函数中确认. 应用该原则, 将消息 2 修改为$\{h(I, R, ni, nr, kir), nr, R, \{kir\}_{ni}\}_{pk(I)}$, 使 *kir*、*nr*、*ni* 在同一个哈希函数中得到确认. 修改后的协议满足 *I*、*R* 的 *Nisynch* 性质.

模型 9: 是模型 5 的变形, 其中 *n* 是由 *R'*产生的随机数. 该模型能保证 *R*、*R'*对 *n* 的 *Commit* 性质. 模型 9 的证明类似模型 5.

2.2 中对 PKMv2-RSA 协议的中间人攻击可用模型 9 解决. 由于该协议要求 *MS*、*BS* 在 *prepak* 上有 *Secret* 性质, 因此也要应用模型 3. 在使用模型 3、9 和共同确认原则后, 协议修改如下:

① $MS \rightarrow BS : \{msrand, \{said\}_{pk(BS)}, MS\}_{sk(MS)}$

② $BS \rightarrow MS : \{h(msrand, said, bsrand, prepak), bsrand, \{\{prepak\}_{said}, MS\}_{pk(MS)}, BS\}_{sk(BS)}$

③ $MS \rightarrow BS : \{h(bsrand, prepak), BS\}_{sk(MS)}$

其中 *msrand*、*said*、*bsrand*、*prepak* 对应于模型 9 中 *n*. 修改后的协议满足 *MS*、*BS* 上全部安全性质.

3.3 Multi-party 模型

Multi-party 模型旨在解决角色混淆攻击. 此类攻击形成的原因较复杂. 中间人攻击、交错攻击都可导致角色混淆. 例如 2.4 中攻击可应用模型 7 解决. 实验中发现一种能预防大部分 multi-party 协议角色混淆攻击的协议模型. 模型描述如下:

模型 10: 假设协议有 p 个参与者, 分别为 $R_0,...,R_{p-1}$. 令 $AR(x) = \{R_0, ... , R_{x-1}, R_{x+1}, ... , R_{p-1}\}$, 则在原协议基础上, 将每个主体 *x* 发送的消息中加入$\{AR(x)\}_{sk(x)}$.

模型 10 是一个基于大量实验而发现的经验性的结论, 它通过对角色信息双向签名, 使 AKC 攻击者无法篡改包含身份信息的消息. 但该模型不能完全保证协议不受到角色混淆攻击. 在实际应用中, 也可以使用$\{h(AR(x))\}_{sk(x)}$以减少效率损失. 模型 10 的应用可参考 multi-party NSL 协议.

在上述讨论中发现, 通过模型 1~10 可以解决第 2 节中提到的大部分攻击. 但 2.3 中 multi-party 协议的交错攻击仍未得到解决. 由于 multi-party 协议的情况比较复杂, 本文未找到一个完全正确的模型来应用在协议上. 但在实验中发现了一个设计原则, 可一定程度避免 AKC 下 multi-party 协议的交错攻击.

共享信息原则: Multi-party 协议中的每个主体, 需要确认其他主体产生过的所有信息. 当某一主体产生的信息需要对另一主体透明时, 可将该信息进行哈希处理再传送. 使用共享信息原则, 可避免由于主体之间信息不对称, 造成攻击者对不同运行之间消息的篡改转发, 从而避免 multi-party 协议的交错攻击.

4 转换算法和实例分析

本节应用第 3 节中结论, 提出 AKCS 协议转换算法. 对于一个受 AKC 攻击的协议, 如果它在 Dolev-Yao 模型下是安全的, 则算法能将其转换为 AKCS 协议; 如果它在 Dolev-Yao 下也受攻击, 则算法能提高其安全性, 但不保证是 AKCS. 这是由于本算法只针对 AKC 攻击问题对协议进行转换, 而对 Dolev-Yao 下也受攻击的协议来说, 它们除 AKC 攻击以外存在其他攻击.

4.1 算法描述

使用 *M* 表示消息, *R* 表示角色, *N* 表示随机数集合, 对算法中部分操作进行如下定义:

1) *Send(M) / Recv(M)*: 表示 *M* 的发送者 / 接收者.
2) *Type(M)*: 表示 *M* 的类型(*send* 或 *recv*).
3) *Next(M)*: 若 *R* 是 *M* 的发送者, 则 *Next(M)*表示在 *M* 后第一条被 *R* 接收的消息.
4) *Fresh(M)*: 表示 *M* 中由发送者最新产生的随机数集合. 例如 BKE 协议中 *Fresh(2) = {nr, kir}*, 即消

息 2 中 *R* 最新产生 *nr*、*kir*.

5) *Conductor(N)*: 表示产生 *N* 的角色集合. 例如 BKE 协议中 *Conductor(ni,nr) = {I,R}*.
6) *SecretClaim(R)*: 表示 *R* 中声明 *Secret* 性质的随机数集合. 假设 R 有声明 *claim(R, Secret, m)*和 *claim(R, Secret, n)*, 则 *SecretClaim(R) = {m, n}*.
7) *CommitClaim(R, N)*: 判断 *R* 中是否有对 *N* 的 *Commit* 性质声明, 返回 *true* 或 *false*.
8) *NiagreeClaim(R) / NisynchClaim(R)*: 判断 *R* 是否有 *Niagree / Nisynch* 性质声明, 返回 *true* 或 *false*.
9) *EncryptionPK(M, N) / EncryptionSK(M, N)*: 判断 *M* 中是否存在对 *N* 的 *pk(recv(M)) / sk(send(M))*加密, 返回 *true* 或 *false*. 例如消息 $I \rightarrow R : \{h(n)\}_{sk(R)}$ 中 *EncryptionSK(M, n) = true*; 消息 $I \rightarrow R : \{k\}_{pk(R)}, \{n\}_k$ 中由于 *pk(R)* 加密 *k* 且 *k* 加密 *n*, 则 *EncryptionPK(M, n) = true*.
10) *SecretModel_n(R, N)*: 表示应用模型 *n* 维护 *claim(R, Secret, N)*性质.
11) *CommitModel_n(R, R', N)*: 表示应用模型 *n* 维护 *claim(R, Commit, R', N)*性质.
12) *RoleInfoModel(R, R')*: 表示对 *R*、*R'*应用模型 10.
13) *HashPrinciple(N)*: 表示对 *N* 应用哈希加密原则.
14) *AfirmPrinciple(M)*: 表示对 *M* 应用共同确认原则.
15) *SharePrinciple(R)*: 表示对 *R* 应用共享信息原则.

算法的主体框架如算法 1 所示.

算法 1. *Protocol_Transform(Protocol p)*

```
1)  Secret_Transform(p)
2)      if   verification succeed
3)              return
4)  Commit_Transform(p)
5)      if verification succeed
6)              return
7)  Multiparty_Transform(p)
```

算法按照 Secret 模型→Commit 模型→Multi-party 模型这 3 步进行协议转换, 当协议在某步骤执行后通过性质验证时, 可停止转换. 算法对协议的转换遵循两个原则: 尽量不破坏原协议的结构, 尽量减少转换所带来的效率损失. 在转换过程中, 如果需要使用哈希函数、公钥、私钥加密, 则尽量使用原协议本身的函数而不增加新的.

Secret 模型的转换算法如算法 2 所示.

算法 2. *Secret_Transform(Protocol p)*

```
1)  for each role R
2)      if   |Conductor(SecretClaim(R))| = 1
3)          for each fresh N in SecretClaim(R)
4)              if Conductor(N) = R
5)                  SecretModel_1(R, N)
6)              else
7)                  SecretModel_2(R, N)
8)      else
9)          SecretModel_4(R, SecretClaim(R))
10)     HashPrinciple(SecretClaim(R))
```

算法对每个 *R* 进行迭代: 如果 *R* 中声明为 *Secret* 的随机数集合是由同一个角色产生的(第 2 行), 则根据这些随机数的来源使用模型 1 或 2 转换; 否则按模型 4 转换. 转换过程中应用哈希加密原则.

Commit 模型的转换算法如算法 3 所示.

算法 3. *Commit_Transform(Protocol p)*

```
1)  for each message M(Type(M) = send) :
2)   for each fresh N in Fresh(M)
3)      if ( N ∉ CommitClaim(Send(M)) or
           NisynchClaim(Send(M)) = false ) and
         ( N ∈CommitClaim(Recv(M)) and
           NisynchClaim(Recv(M)) = true )
4)              CommitModel_5(Send(M), Recv(M), N)
5)      else if Fresh(Next(M)) = ∅
6)          if EncryptionPK(M, N) = true
7)              CommitModel_6(Send(M),Recv(M), N)
8)          else
9)              CommitModel_9(Send(M),Recv(M), N)
10)     else if EncryptionPK(M, N) = true
11)         CommitModel_7(Send(M),Recv(M), N)
12)     else if EncryptionSK(M, N) = true
13)         CommitModel_9(Send(M),Recv(M), N)
14)     else if EncryptionPK(Next(M), N) = true
15)         CommitModel_8(Send(M),Recv(M), N)
16)     else
17)         CommitModel_9(Send(M),Recv(M), N)
18)     AfirmPrinciple(Next(M))
```

算法对消息按发送顺序迭代, 对消息中由发送者新产生的随机数 N 进行 Commit 模型转换: (1)若发送

方没有 *Commit(N)*或 *Nisynch* 性质声明而接收方有(第 3 行), 则按模型 5 转换; (2)若下一条消息中没有新产生的随机数(第 5 行), 则根据 *N* 的加密形式按模型 6 或 9 转换(*Next(M)*是模型 6 中消息 1, 模型 9 中消息 2) ; (3)若 *N* 存在公钥加密(第 10 行), 则按模型 7 转换(*Next(M)* 是模型 7 中消息 1) ; (4)若 *N* 存在私钥加密(第 12 行), 则按模型 9 转换; (5)若 *N* 在下一条消息中存在公钥加密(第 14 行), 则按模型 8 转换(*Next(M)*是模型 8 中消息 1); (6)以上条件都不成立时按模型 9 转换. 转换过程中应用共同确认原则.

Multi-party 模型的转换算法如算法 4 所示:

算法 4. *Multiparty_Transform(p)*

1) for each role *R1*
2) for each role *R2*
3) *RoleInfoModel(R1, R2)*
4) *SharePrinciple(R2)*
5) *SharePrinciple(R1)*

算法对每一对角色按照模型 10 转换. 转换过程中应用共享信息原则.

4.2 实例演示

第3节中给出过基于模型来转换协议的示例. 4.1 中算法本质上也是在不同条件下对消息应用不同模型. 下面以 Kerberos 协议为例, 说明算法过程.

Kerberos[9]是 Windows 系统中用于身份认证的协议, 能够使用户只输入一次身份验证信息就可以访问多个服务. 协议由三部分组成: (1)用户 *C* 与认证服务器 *KAS* 交互并获得会话密钥 *AK*; (2)用户与许可证颁发服务器 *TGS* 会话并获得服务凭证 *ST*; (3)用户使用 *ST* 向网络服务发送请求. Kerberos 有一个公钥加密版本 PKINIT, 该版本修改了协议的第一部分, 如下所示:

① $C \rightarrow KAS : CERT(C), \{Tc, n2\}_{sk(C)}, C, TGS, n1$

② $KAS \rightarrow C : \{CERT(KAS), \{k, CK\}_{sk(KAS)}\}_{pk(C)}, C, TGT, \{AK, n1, Tk, TGS\}_k$

$CERT(X) = \{X, pk(X)\}_{sk(CA)}$, $CK = h(CERT(C), \{Tc, n2\}_{sk(C)}, C, TGS, n1)$, $TGT = \{C, AK\}_{k(KAS, TGS)}$. 其中 *CA* 是认证中心, *Tc(Tk)*是 *C(KAS)*的时间戳, *n1*、*n2* 是 *C* 产生的随机数, *k*、*AK* 是 *KAS* 产生的随机数. 客户 *C* 通过上述交互获取 *AK*, 并利用 *AK* 进行后续与 *TGS* 的会话. 因此, PKINIT 协议要求会话密钥的保密性以及数据的完整性, 即存在声明: *claim(C, Secret, AK)*、*claim(C, Nisynch)*、*claim(K, Secret, AK)*、*claim(K, Nisynch)*. 需要说明的是, 由于 *TGS* 没有参与第一部分会话, 因此 *k(KAS,TGS)*不影响验证结果.

使用 Scyther[3]工具可验证上述协议在 AKC 攻击下不满足 *Secret* 和 *Nisynch* 性质. 因此需要进行转换.

根据算法首先应用 *Secret* 模型, 将消息 1 中 *n1* 用 *pk(KAS)*加密, 消息 2 中 *k* 用 *n1* 加密. 修改如下:

① $C \rightarrow KAS : CERT(C), \{Tc, n2\}_{sk(C)}, C, TGS, \{n1\}_{pk(KAS)}$

② $KAS \rightarrow C : \{CERT(KAS), \{\{k\}_{n1}, CK\}_{sk(KAS)}\}_{pk(C)}, C, TGT, \{AK, n1, Tk, TGS\}_k$

其次应用 Commit 模型. 过程如下:

1) 处理消息 1: *n1*、*n2*、*Tc* 是消息 1 中 *C* 最新产生的随机数, 且 *n1* 在消息 1、2 中都有公钥加密, *n2*、*Tc* 在消息 1、2 中都有私钥加密. 由算法知 *n1*、*n2*、*Tc* 使用模型 9 确认(*n1* 也可用模型 8 确认, 但使用模型 9 对原协议改动更小). 由于 *CK* 已满足模型 9, 只需将消息 1 中 *ni* 移入 *sk(C)*内.
2) 处理消息 2: 由于消息 2 在第一部分中不存在下一条消息, 根据算法使用模型 5 确认.

根据 1)、2)修改后的协议如下:

① $C \rightarrow KAS : CERT(C), \{Tc, n2, \{n1\}_{pk(KAS)}\}_{sk(C)}, C, TGS$

② $KAS \rightarrow C : \{CERT(KAS), \{\{k\}_{n1}, CK\}_{sk(KAS)}\}_{pk(C)}, C, TGT, \{AK, n1, Tk, TGS\}_k$

最后应用 Multi-party 模型, 在消息 1 中加入$\{K, TGS\}_{sk(C)}$信息, 而消息 2 中由于$\{k, CK\}_{sk(KAS)}$已包含$\{C, TGS\}_{sk(KAS)}$, 因而不需修改. 最终修改后的协议如下:

① $C \rightarrow KAS : CERT(C), \{Tc, n2, C, TGS, \{n1\}_{pk(KAS)}\}_{sk(C)}$

② $KAS \rightarrow C : \{CERT(KAS), \{\{k\}_{n1}, CK\}_{sk(KAS)}\}_{pk(C)}, C, TGT, \{AK, n1, Tk, TGS\}_k$

使用 Scyther 工具可验证, 在 AKC 模型下, 修改后的协议满足 *Secret* 和 *Nisynch* 安全性质.

4.3 效果分析

实验使用 Scyther 工具发现电子商务、电子邮件等协议中的 AKC 攻击. 应用算法对协议进行了转换, 并对转换后协议的安全性质进行验证. 转换算法对协议安全性质的影响如表 1, 对协议效率的影响如表 2.

表 1 中 D、O、T 分别表示协议在 Dolev-Yao 模型下、AKC 模型下和经过转换后在 AKC 模型下的验证结果. √表示满足性质, ×表示不满足性质, －表示没有此性质声明. 表中展示了对 Secret、Niagree、Nisynch 三个性质的验证结果. 可以看出, 除 Tmn、Splice 协议外, 算法对协议的转换能够保证 AKC 下安全性质成立.

而 Tmn、Splice 在 Dolev-Yao 模型下存在攻击，说明算法对此类协议不能保证转换成 AKCS.

表 2 展示了协议转换前、后公(私)钥总数目的变化. 在 PKI 体系下加密和解密占据一定时间，因此对协议转换应尽量减少公(私)钥的使用. 表 2 说明，除 Tmn 和 3-party 协议外，转换算法对协议中公(私)钥的增加维持在 3 个以内，并且多数协议中没有增加，因此对效率的影响较小.

表 1 转换算法对安全性质的影响

协议[5]-[13]	Secret			Niagree			Nisynch		
	D	O	T	D	O	T	D	O	T
BKE	√	×	√	√	×	√	√	×	√
multi-party BKE	√	×	√	√	×	√	√	×	√
NSL	√	×	√	√	×	√	√	×	√
multi-party NSL	√	×	√	√	×	√	√	×	√
CCITT X.509(1)	√	×	√	√	×	√	√	×	√
CCITT X.509(3)	√	×	√	√	×	√	√	×	√
ISOIEC9798-3-5	—	—	—	√	×	√	√	×	√
Kerberos	√	×	√	√	×	√	√	×	√
PKMv2-RSA	√	×	√	√	×	√	√	×	√
SET register	—	—	—	√	×	√	√	×	√
SET purchase	—	—	—	√	×	√	√	×	√
SPLICE/AS	√	×	√	×	×	×	×	×	×
TLS	√	×	√	—	—	—	—	—	—
Tmn	×	×	√	×	×	×	×	×	×
Email	—	—	—	×	×	√	×	×	√

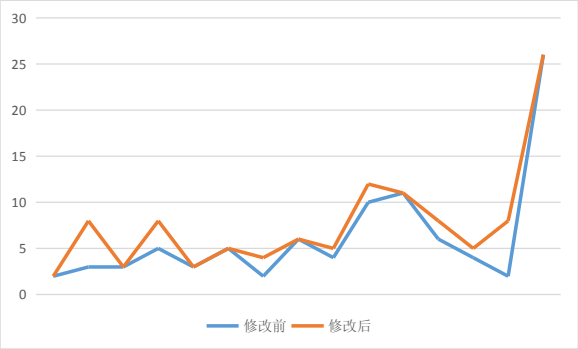

表 2 转换算法对公钥、私钥数目的影

5 结论

本文给出了针对 AKC 攻击的协议转换算法，实验表明算法具备可行性. 本文提出的 AKCS 协议模型和设计原则可以有效抵制 AKC 攻击，在攻击者能力越来越强的网络环境中，可以作为协议设计的参考.

参考文献

1 Blake-Wilson S, Johnson D, Menezes A. Key agreement protocols and their security analysis. 6th IMA International Conference on Cryptography and Coding. Cirencester, UK. 1997. 30–45.

2 Basin D, Cremers C, Horvat M. Actor key compromise: consequences and countermeasures. 2014 IEEE Computer Security Foundations Symposium. Vienna, Austria. 2014. 244–258.

3 Cremers CJF. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. 2008 International Conference on Computer Aided Verification. Princeton, USA. 2008. 414–418.

4 Cremers C, Mauw S. Operational Semantics and Verification of Security Protocols. Springer Berlin Heidelberg, 2012.

5 Blom S, Groote J F, Mauw S, Serebrenik A. Analysing the BKE-security Protocol with μCRL. Electronic Notes in Theoretical Computer Science, 2005, 139(1): 49–90.

6 Andova S, Cremers C, Gjøsteen K, Mauw S, Mjølsnes SF. A framework for compositional verification of security protocols. Information & Computation, 2006, 206(24): 425–459.

7 Hwang T, Chen YH. On the security of SPLICE/AS — The authentication system in WIDE Internet. Information Processing Letters, 1995, 53(2): 97–101.

8 Lowe G. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. Second International Workshop on Tools & Algorithms for Construction & Analysis of Systems. Springer-Verlag. 1996: 147–166.

9 Cervesato I, Jaggard AD, Scedrov A, Walstad C. Breaking and fixing public-key Kerberos. Proc. of the 11th Asian computing science conference on Advances in computer science: secure software and related issues. Springer-Verlag, 2006: 402–424.

10 Bella G, Massacci F, Paulson LC, Tramontano P. Formal Verification of Cardholder Registration in SET. Lecture Notes in Computer Science, 2001, 1895: 159–174.

11 Bella G, Massacci F, Paulson LC. An overview of the verification of SET. International Journal of Information Security, 2005, 4(4): 17–28.

12 Dierks T, Rescorla E. The transport layer security (TLS) protocol version 1.2, IETF RFC, August 2008.

13 Galdi C, Giordano R. Certified e-mail with temporal authentication: An improved optimistic protocol. Trust and Privacy in Digital Business. Springer Berlin Heidelberg. 2004: 181–190.